GEORGIA INSTITUTE OF TECHNOLOGY

School of Electrical and Computer Engineering

ECE-6276-A DSP Hardware System Design (Fall 2018)

Lab 3: Multiplier Design

Assigned Date : Wednesday, September 12, 2018

Due Date : 23:55, Thursday, September 20, 2018

---

1 Introduction

In image processing and Convolutional Neural Networks, there are a lot of multiplications and additions. The multiplication usually applies a 3×3 mask and move it around the whole image to get the dot production. For a high speed digital design, we usually parallelize the multiplication in either way showed in the Figure 1 below.[1] In both way, both multiplication share the same coefficient as one of the 2 operands.

In this lab, we will be designing 2 multipliers **for signed operations**, e.g. $A \times B$ and $D \times B$, where B is the common coefficient. A, B, D are all 8-bit signed fixed point numbers. Then we will update the design in a more area-efficient way.

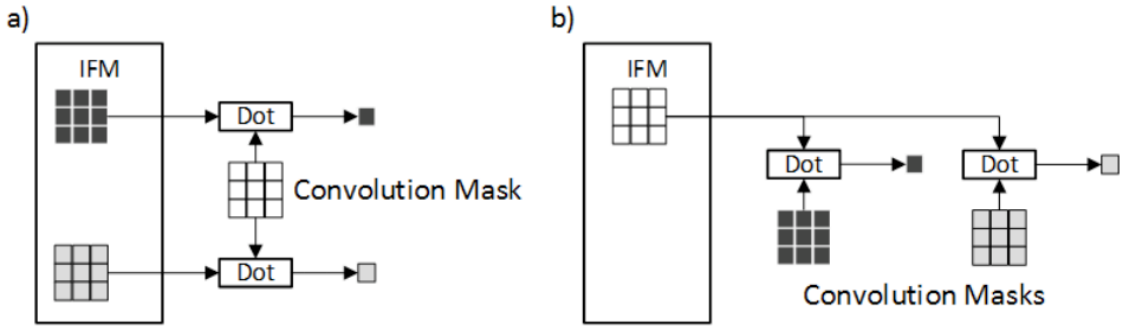

Figure 1 Parallelism in INT8 Convolution

1.1 2-Stage Pipelined Multiplier Design

In this step, we will design 2 multipliers in a 2-stage pipeline (See Figure 2). The inputs are stored into registers in the 1st stage and we do multiplications on the 2nd stage.

[1] Xilinx: 8-bit Dot-Product Acceleration. `https://pdfs.semanticscholar.org/3ac6/4259d37ad76c640333bf8cfccd36bb9bc4f0.pdf`

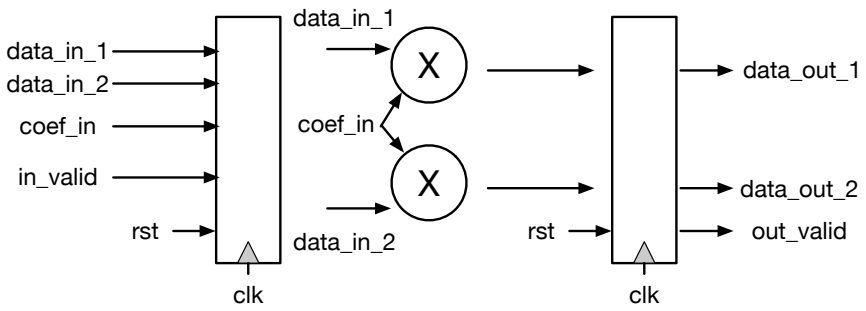

Figure 2 Block diagram of 2 multipliers

1.2 Updated 3-Stage Pipelined Multiplier Design

There are limited resources (LUTs and DSPs) on the FPGA. If we design every MAC unit (Multiplier and Accumulator) separately in a large design project (although we are not designing accumulator in this lab), we will soon run out of resources. Since the DSP48E1[2] we are using supports 25×18 bit multiplications, we concatenate 2 operands `data_in_1` and `data_in_2` together into one longer-bit operand. Figure 3 shows how the concatenation and splitting work for a packed word.[3] In this step, we will design multiplier using a pakced word following Figure 3.

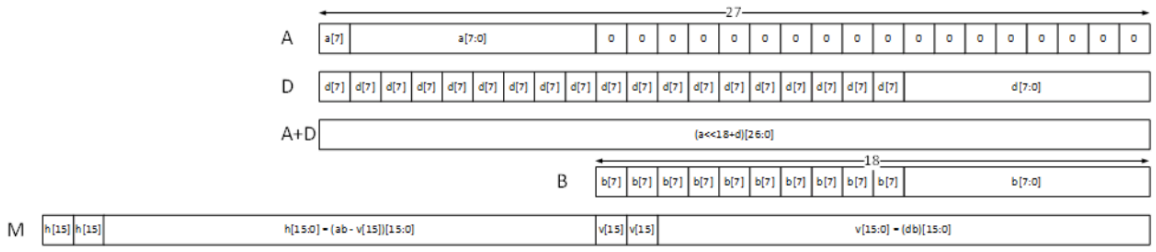

Figure 3 Two products in a packed word

The block diagram of this 3-stage pipelined multiplier is showed in Figure 4.

Note: Lab 4 will be based on this 3-Stage Pipelined Multiplier Design. So please follow this block diagram.

[2]For more information, please refer to `https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf`

[3]Xilinx: 8-bit Dot-Product Acceleration. `https://pdfs.semanticscholar.org/3ac6/4259d37ad76c640333bf8cfccd36bb9bc4f0.pdf`

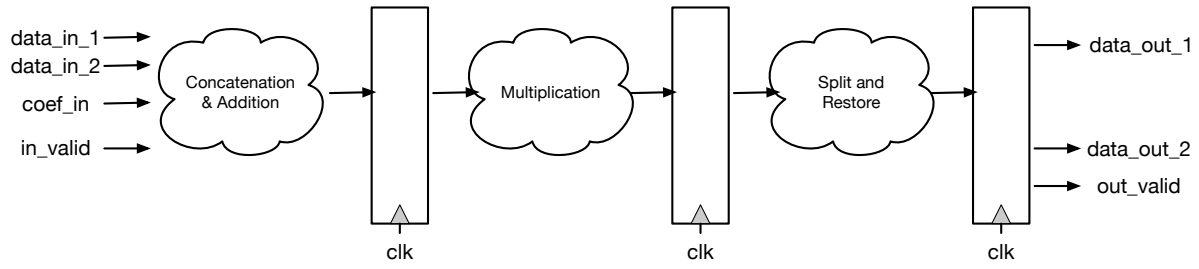

Figure 4 Block diagram of updated multipliers

2 Instructions

1. Complete your 2-multipliers design `./src/multiplier.vhd` for Section 1.1 (Template is given. Do not modify the ports).
2. Run the simulation using the testbench `./tb/tb_multiplier.vhd` and compare your output file `./run/output.txt` with the reference output file `./run/output_ref.txt` until they match exactly the same.
3. Run the synthesis and implementation of your design `multiplier.vhd` in Vivado using the provided constraint file `./constraints.xdc`.
4. Complete your updated multiplier design `./src/multiplier_updated.vhd` for Section 1.2 (Template is given. Do not modify the ports).
5. Run the simulation using the testbench `./tb/tb_multiplier_updated.vhd` and compare your output file `./run/output_updated.txt` with the reference output file `./run/output_updated_ref.txt` until they match exactly the same.
6. Run the synthesis and implementation of your design `multiplier_updated.vhd` in Vivado using the provided constraint file `./constraints.xdc`.

3 Deliverables

1. Create a PDF report containing:
   - Tables of
     (a) Resources (Only "Slice Logic", "DSP", "IO and GT Specific", "Primitives") from `multiplier_utilization_placed.rpt` and `multiplier_updated_utilization_placed.rpt`
     (b) Power (Dynamic and Static) from `multiplier_power_routed.rpt` and `multiplier_updated_power_routed.rpt`
     (c) Worst Negative Slack ("Design Timing Summary") from `multiplier_timing_summary_routed.rpt` and `multiplier_timing_summary_routed.rpt`
   - Answer the questions:

(a) (10% of grades) Compare the tables above and briefly explain the difference between two designs. (Answer should not exceed 4 sentences)

(b) (10% of grades) What is the advantage of using DSP instead of LUTs on FPGA? (Answer should not exceed 4 sentences)

The name of the PDF should be of the form `lab3_firstname_lastname.pdf` , e.g. `lab3_george_burdell.pdf` .

2. (25% of grade) The completed design file `multiplier.vhd` .
3. (45% of grade) The completed design file `multiplier_updated.vhd` .
4. The simulated output file `output.txt` and `output_updated.txt` in the "run" folder.
5. (10% of grade) The implemented area (utilization), power and timing reports, namely,

   `multiplier_utilization_placed.rpt`

   `multiplier_power_routed.rpt`

   `multiplier_timing_summary_routed.rpt`

   `multiplier_updated_utilization_placed.rpt`

   `multiplier_updated_power_routed.rpt`

   `multiplier_updated_timing_summary_routed.rpt`

Move all of these files into a folder called `lab3_firstname_lastname_gtID` . Zip the folder and upload the archive `lab3_firstname_lastname_gtID.zip` on T-Square (eg. `lab3_george_burdell_123456789.zip`). **Please strictly follow this naming convention. Otherwise my script will not work and you might get points deduction.**

Note: Late submissions are not accepted. In case of extraordinary circumstances, written permission must be obtained from Dr. Madisetti.

4 Appendix

What does Synthesis and Implementation mean for FPGA?

Synthesis is the process of converting an RTL into its gate (component) level representation. For e.g.: A Multiplexer written in VHDL with case statements is converted to its gate level logic (using gates). The gate-level logic is also called netlist. The Programmable Logic of an FPGA has a basic element called Logic Cell (Slice). Each Logic Cell has a Look-up Table (LUT), Flip-Flops and multiplexer. Since a mux can be used to map any digital logic, it is possible to map any netlist to these resources on FPGA. Usually the combinational logic is mapped into LUTs on FPGA. This is called Implementation. In addition to Logic Cells, FPGA also has Block RAMs (for memories), DSP elements (optimized adders and multipliers) and IO ports.